# ЧАСТЬ IV

# Безопасность

Выбор правильной стратегии безопасности — ключевая часть любого распределенного приложения, особенно если речь идет о крупномасштабном веб-приложении, доступ к которому предоставляется через Интернет. В этой книге вы найдете не менее восьми глав, описывающих средства безопасности ASP.NET.

В главе 19 мы начнем с высокоуровневого обзора трех основ безопасности: аутентификации, авторизации и конфиденциальности. Исходя из этой перспективы, вы будете готовы к рассмотрению двух ключевых систем аутентификации пользователей в ASP.NET: аутентификации с помощью форм (глава 20), которая представляет простую, но гибкую платформу защиты публичных веб-сайтов, и аутентификации Windows (глава 22), использующей существующие учетные записи Windows для аутентификации пользователей, которая наиболее широко распространена в корпоративных сетях. Вы также познакомитесь с высокоуровневыми службами безопасности ASP.NET, такими как членство, роли и профили. Членство (глава 21) предоставляет встроенные службы безопасности и позволяет ASP.NET управлять базой данных заднего плана, хранящей мандаты пользователей. Роли (глава 24) позволяют помещать пользователей в логические группы, которым затем могут быть выданы различные привилегии. Профили (глава 25) дают возможность хранить специфическую для каждого пользователя информацию в базе данных серверной стороны, без написания собственного кода ADO.NET. Хотя это мощные средства, “за кулисами” они влекут за собой множество деталей. Чтобы правильно настроить их работу, вы должны построить специального поставщика — эта тема затрагивается в главе 26.

И, наконец, в главе 25 предлагается экскурс в криптографические средства .NET, которые важны для защиты ценной информации перед ее сохранением в базе данных. В отличие от других средств безопасности, описанных в этой части, применение криптографических классов .NET не ограничено ASP.NET, хотя они часто оказываются полезными в веб-приложениях, позволяя реализовать такие вещи, как построение устойчивых к взлому строк запросов.

ГЛАВА 19

# Модель безопасности ASP.NET

Безопасность — важнейшая часть веб-приложений, и она должна приниматься во внимание с первого этапа процесса разработки. По сути, безопасность — это все, что касается защиты вашего имущество от неавторизованных действий. И для ее обеспечения используется несколько механизмов, включая идентификацию пользователей, выдачу или отзыв прав доступа к важным ресурсам, а также защиту информации, хранящейся на сервере и передающейся по сети. Во всех этих случаях необходима некая фундаментальная платформа, обеспечивающая базовую функциональность безопасности. ASP.NET удовлетворяет эту потребность благодаря встроенным средствам, которые можно применять для обеспечения защиты своих приложений.

Платформа безопасности ASP.NET включает классы для аутентификации и авторизации пользователей, а также для обращения с аутентифицированными пользователями в приложениях. Она также включает высокоуровневую модель для управления пользователями и ролями, как программно, так и с помощью инструментов администрирования. Более того, платформа .NET Framework сама по себе предоставляет набор базовых классов для обеспечения конфиденциальности и целостности через шифрование и цифровые подписи.

В настоящей главе предложен общий путеводитель по средствам безопасности ASP.NET. В последующих главах вы углубите свои познания по каждой теме из числа представленных здесь. А пока мы проведем краткое представление ключевых средств обеспечения безопасности .NET. Более важно, что вы получите общее представление о том, как можно встроить средства безопасности в создаваемую программную архитектуру и проектное решение, и увидите, какие факторы наиболее важны при создании безопасного программного обеспечения.

## Что означает создание безопасного программного обеспечения

Хотя платформа безопасности, предложенная .NET и ASP.NET, достаточно мощная, все же стоит постоянно иметь в виду базовые принципы и правильно применять эти средства в нужный момент. В слишком многих проектах забота о безопасности проявляется запоздало; архитекторы и разработчики не думают о ней на ранних стадиях проекта. Но если не принять во внимание безопасность с самого начала, а именно — при разработке проектного решения и архитектуры приложения, — то как можно будет правильно и вовремя воспользоваться средствами защиты, предлагаемыми .NET Framework?

Таким образом, важно учитывать вопросы обеспечения безопасности с первого момента работы. Это единственный способ принятия правильных решений, касающихся защиты, в процессе разработки архитектуры и проектного решения.

## Понятие потенциальных угроз

Создание безопасной архитектуры и проектного решения требует глубокого понимания среды приложения. Построить безопасное программное обеспечение не удастся, если не известно, кто имеет доступ к приложению и где находятся уязвимые места для атак. Таким образом, наиболее важный фактор для создания безопасной программной архитектуры и проектного решения заключается в хорошем понимании таких факторов среды, как пользователи, точки входа и потенциальные угрозы с точками для атаки.

Именно поэтому *моделирование угроз* становится все более важным в современном процессе разработки программного обеспечения. Моделирование угроз — это структурированный способ анализа среды приложения с точки зрения возможных опасностей, классификация угроз и решение относительно приемов их смягчения. При таком подходе решения относительно технологий безопасности (таких как аутентификация и SSL-шифрование) всегда имеет действительное основание — потенциальную угрозу.

Однако моделирование угроз важно по еще одной причине. Как вы, возможно, знаете, не все потенциальные угрозы могут быть смягчены применением технологий защиты, такими как аутентификация и авторизация. Другими словами, некоторые из них вообще невозможно разрешить технически. Например, банковское онлайновое решение может использовать SSL для защиты трафика веб-сайта. Но как пользователи могут знать, что они действительно используют банковскую страницу, а не хакерский поддельный веб-сайт? Хорошо, единственный способ убедиться в этом — проверить сертификат, используемый для установки канала SSL. Но пользователи должны быть предупреждены об этом, а потому вы должны каким-то образом их информировать. Поэтому "техника смягчения" угроз — это не только технологии защиты. Это включает требование того, чтобы все пользователи знали, как проверить сертификат. (Конечно, вы не можете заставить их это делать, но если система спроектирована соответствующим образом, все же можно стимулировать к этому большинство из них.) Моделирование угроз — метод анализа, помогающий выявить обстоятельства вроде этих, а не только факторы технического порядка.

---

**На заметку!** Моделирование угроз — обширная тема, которая выходит за рамки настоящей книги. Если интересуетесь, можете обратиться к множеству книг, включая *Writing Secure Code, Second Edition* (Microsoft Press, 2002 г.), а также *Threat Modeling* (Microsoft Press, 2004 г.). Вдобавок книга *Security Development LifeCycle* (Microsoft Press, 2006 г.) исключительно полезна для менеджеров проектов и архитекторов. Эта книга сосредоточена на обеспечении безопасности как неотъемлемой части цикла разработки программного обеспечения — начиная с первоначального планирования и включая построение архитектуры, разработку, тестирование и сопровождение. В ней показано, как управление проектами Microsoft обеспечивает безопасность в качестве неотъемлемой части проекта, причем гладким и прагматичным способом.

---

## Правила безопасного кодирования

Разумеется, только безопасная архитектура и проектное решение не могут сделать приложение абсолютно защищенным. Это лишь один из наиболее важных факторов. После разработки безопасной архитектуры и проектного решения понадобится также написать безопасный код.

При написании кода веб-приложений всегда имейте в виду следующие правила.

- *Никогда не доверяйте пользовательскому вводу.* Предполагайте, что каждый пользователь является злоумышленником, пока он не докажет обратное. Таким образом, всегда строго проверяйте пользовательский ввод. Разрабатывайте свой код проверки достоверности так, чтобы он проверял ввод только правильных значений, а не неправильных (неправильных значений всегда больше, чем вы можете себе представить во время разработки приложения).

- *Никогда не используйте конкатенацию строк для формирования операторов SQL.* Всегда применяйте параметризованные операторы, чтобы приложение не было уязвимо для атак внедрением SQL, как было описано в главе 7.

- *Никогда не выводите данные, введенные пользователем, на веб-страницу перед их проверкой и кодированием.* Пользователь может ввести некоторые фрагменты кода HTML (например, сценарии), которые инициируют межсайтовую сценарную уязвимость. Поэтому всегда используйте `HttpUtility.HtmlEncode()` для отмены специальных символов вроде < или > перед выводом их на страницу или применяйте веб-элемент управления, который выполняет такое кодирование автоматически.

- *Никогда не размещайте важные данные, критичную для бизнеса информацию либо данные, касающиеся внутренних правил безопасности, в скрытых полях веб-страницы.* Скрытые поля могут быть легко изменены простым просмотром исходного кода веб-страницы, модификацией и сохранением в файле. Затем злоумышленник просто может отправить локально модифицированную копию веб-страницы на сервер. Существуют подключаемые модули браузеров, которые упрощают такой прием, делая его не сложнее отправки электронной почты в Microsoft Outlook.

- *Никогда не сохраняйте важные или критичные для бизнеса данные в состоянии представления.* Состояние представления — это просто еще одно скрытое поле на веб-странице, и оно может быть легко декодировано и просмотрено. Шифрование состояния представления (как описано в главе 6) помогает защитить информацию, ценную только в течение ограниченного интервала времени, но имейте в виду, что даже шифрованные данные однажды могут быть взломаны, если у злоумышленника есть достаточно времени, ресурсов и мотивации.

- *Включайте SSL при использовании базовой (Basic) аутентификации или аутентификации с помощью форм ASP.NET.* Аутентификация с помощью форм описана в главе 20. Об SSL мы поговорим позднее в настоящей главе, в разделе "Что собой представляет SSL".

- *Защищайте свои cookie-наборы.* Всегда защищайте свои cookie-наборы аутентификации при использовании аутентификации с помощью форм и устанавливайте таймауты насколько возможно короткими, и не длиннее, чем это действительно необходимо.

- *Применяйте SSL.* В общем случае, если веб-приложение обрабатывает важные данные, защищайте весь веб-сайт с помощью SSL. Не забывайте защищать даже каталоги с графическими изображениями или другими файлами, которые не управляются приложением напрямую через SSL.

Конечно, это лишь несколько общих важных моментов. Чтобы получить общую картину ситуации для конкретного приложения, необходимо разработать модель угроз, чтобы составить полный список потенциальных опасностей. Вдобавок вкладывайте средства в постоянное обучение разработчиков, потому что хакерские приемы и технологии также не стоят на месте, как и все прочие технологии.

Если вы пренебрегаете хотя бы одним из приведенных правил, то все прочие средства защиты становятся в большей или меньшей мере бесполезными. Никогда не забывайте следующий принцип: система защиты ровно настолько надежна, насколько надежна ее самая слабая часть.

## Понятие стража

Хороший способ повысить степень безопасности приложения — размещать компоненты в таком месте, которое требует защиты. Концептуальный шаблон *стража* (gatekeeper) применяет модель конвейера к организации инфраструктуры безопасности. Эта модель помогает укрепить безопасность.

Модель стражей предполагает, что безопасное приложение всегда имеет больше механизмов защиты, чем это необходимо. Каждый из этих механизмов реализован в виде стража, отвечающего за проверку некоторых условий защиты. Если один из таких стражей не сработает, то атакующий столкнется со следующим стражем в конвейере. И чем больше стражей имеется в вашем приложении, тем труднее приходится злоумышленнику. На самом деле эта модель отвечает ключевому принципу создания безопасных приложений: обеспечивать насколько возможно высокую степень защиты и создавать максимум проблем нарушителям.

На рис. 19.1 показан конвейер стражей. В конце этого конвейера можно видеть защищенный ресурс (которым может быть что угодно, даже ваш собственный код страницы). Защищенный ресурс будет доступен или выполнен только в том случае, если все стражи откроют к нему доступ. Если хотя бы один из них откажет в доступе, обработка запроса будет прекращена и клиент получит исключение, связанное с безопасностью.

Реализация центрального механизма безопасности в такой манере — вообще хорошая идея. Точно так же вы можете защитить бизнес-уровень своего приложения. Инфраструктура приложений ASP.NET также применяет этот механизм. Среда ASP.NET включает несколько стражей, каждый из которых проверяет несколько условий и таким образом защищает ваше приложение. В последующих разделах настоящей главы вы ознакомитесь со стражами, которые включает в себя платформа ASP.NET, и с зоной ответственности каждого из них.

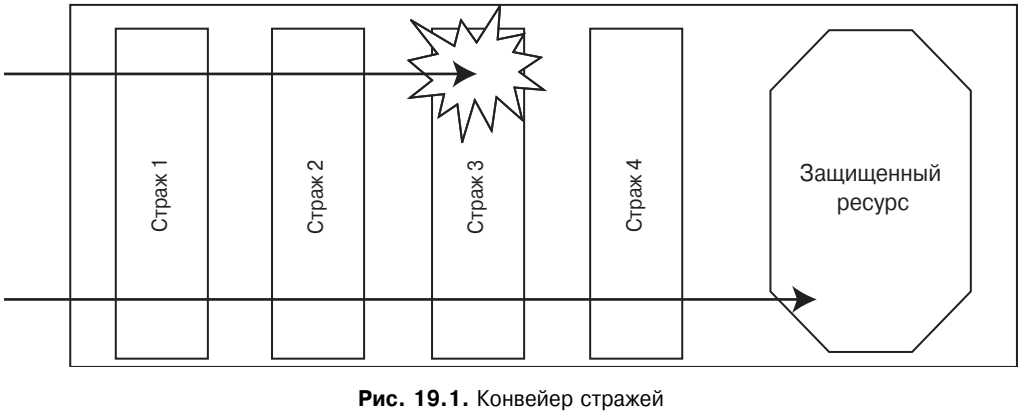


**Рис. 19.1.** Конвейер стражей

# Понятие уровней безопасности

В основном для большей части веб-приложений основные задачи для реализации защиты (помимо идентифицированных во время моделирования угроз) всегда одни и те же.

- *Аутентификация*. Прежде всего, вы должны аутентифицировать пользователей. Аутентификация задает вопрос: кто идет? В конечном итоге она определяет, кто работает с вашим приложением на другой стороне.
- *Авторизация*. Теперь, когда известно, кто работает с приложением, понадобится решить, какие операции данный пользователь может выполнять и к каким ресурсам обращаться. Другими словами, авторизация отвечает на вопрос: каков уровень допуска?
- *Конфиденциальность*. Когда пользователь работает с приложением, вы должны гарантировать, что никто другой не сможет видеть важные данные, которые он обрабатывает. Таким образом, необходимо шифровать канал между браузером клиента и веб-сервером. Более того, возможно, придется шифровать данные, сохраняемые в базе данных (или в форме cookie-наборов на стороне клиента), чтобы даже администратор базы данных или другой персонал компании не мог видеть эти данные.
- *Целостность*. И, наконец, вы должны гарантировать, что данные, передаваемые между клиентом и сервером, не изменяются в результате неавторизованного вмешательства. Снизить уровень этой угрозы позволяют цифровые подписи.

ASP.NET включает базовую инфраструктуру для выполнения аутентификации и авторизации. Библиотека базовых классов .NET Framework включает ряд классов в пространстве имен `System.Security`, которые предназначены для шифрования и подписи данных. Более того, SSL — стандартизированный способ обеспечения конфиденциальности и целостности данных, передаваемых между клиентским браузером и веб-сервером. Давайте рассмотрим каждую из этих концепций более подробно.

## Аутентификация

*Аутентификация* — процесс определения идентичности пользователя и обеспечения гарантий этой идентичности. Процесс аутентификации аналогичен регистрации участников конференции. Во-первых, вы предъявляете некоторое свидетельство, доказывающее вашу идентичность (вроде паспорта или водительских прав). Во-вторых, как только идентичность проверена по этой информации, вы получаете личный значок участника конференции, или *маркер*, который постоянно носите с собой на протяжении всей конференции. Любой, кто вас встретит на конференции, сможет легко определить вашу идентичность, взглянув на этот значок, который обычно содержит базовую идентифицирующую информацию, такую как имя и фамилия. Весь этот процесс — пример аутентификации. Как только идентичность установлена, маркер подтверждает ее, так что куда бы вы ни пошли в пределах конкретной области, ваша личность будет известной.

В приложениях ASP.NET аутентификация реализуется одной из следующих возможных аутентифицирующих систем:

- аутентификация Windows;
- аутентификация с помощью форм;
- специальный процесс аутентификации.

В каждом случае пользователь предъявляет некоторое удостоверение при регистрации в системе. Идентичность пользователя отслеживается разными способами, в зависимости от типа аутентификации. Например, операционная система Windows использует 96-битное число, называемое SID (security identifier — идентификатор безопасности) для идентификации каждого входящего пользователя. При аутентификации с помощью форм ASP.NET (которая подробно рассматривается в главе 20) пользователю выдается аутентифицирующий мандат формы, представляющий собой комбинацию значений, которые шифруются и помещаются в cookie-набор.

Вся аутентификация позволяет приложению идентифицировать, какой пользователь присылает каждый запрос. Это хорошо работает для персонализации и пользовательской настройки, потому что вы можете использовать идентифицирующую информацию для выдачи специфичных для пользователя сообщений на веб-странице, изменять внешний вид веб-сайта, добавлять специальное содержимое на основе предпочтений конкретного пользователя и т.п. Однако аутентификации самой по себе недостаточно для ограничения задач, которые разрешено выполнять пользователю на основе его идентичности. Для этого нужна авторизация, о которой речь пойдет ниже. Прежде чем переходить к рассмотрению авторизации, следует взглянуть на заимствование прав, которое тесно связано с аутентификацией.

### Заимствование прав

*Заимствование прав* (impersonation) — это процесс выполнения кода в контексте (или от имени) другого пользователя. По умолчанию код ASP.NET выполняется от имени фиксированной, специфичной для конкретной машины, пользовательской учетной записи (обычно Network Service в IIS 7.*x*). Чтобы выполнить код с применением другой идентичности, можно воспользоваться встроенными в ASP.NET возможностями заимствования прав. Можно применить предопределенную пользовательскую учетную запись либо предположить пользовательскую идентичность, если этот пользователь уже был аутентифицирован посредством учетной записи Windows.

Одна из причин, по которым может применяться заимствование, состоит в возможности использования существующих учетных записей Windows с их привилегиями. Например, рассмотрим приложение, которое извлекает информацию из различных файлов, уже имеющих специфические для пользователей или групп наборы прав. Вместо того чтобы кодировать логику авторизации в своем приложении ASP.NET, можно воспользоваться заимствованием, полагаясь на идентичность конечного пользователя. Таким образом, Windows может выполнить авторизацию за вас, проверяя привилегии при попытке обратиться к файлу. Можно даже включать заимствование только на краткий период времени вместо полного запроса. Подробнее об этом вы узнаете в главе 22.

## Авторизация

*Авторизация* — это процесс определения прав и ограничений, назначенных аутентифицированному пользователю. Если взять аналогию с конференцией, то авторизация — это процесс выдачи допуска на определенные мероприятия, например, на главный доклад. На большинстве конференций можно подписаться на разные уровни доступа — на полный доступ, только вступительное заседание или только на посещение выставочного зала. Это значит, что если вы захотите посетить главное заседание "Конференции профессиональных разработчиков Microsoft", чтобы послушать, что скажет Билл Гейтс, то должны для этого иметь соответствующие права доступа. Как только вы войдете в зал заседаний, сотрудник службы безопасности проверит ваш бейдж участника конференции. На основании указанной на нем информации он либо пропустит вас, либо скажет, что вы не можете войти. Это пример авторизации. В зависимости от

идентифицирующей информации, доступ к запрашиваемым ресурсам либо открывается, либо закрывается.

Пример с конференцией представляет собой случай *авторизации на основе ролей* — когда авторизация определяется правами группы, к которой принадлежит пользователь, а не на том, кто он такой. Другими словами, вы авторизуетесь для доступа в зал заседаний на основании роли (типа допуска), а не вашей специфичной идентифицирующей информации (имени и фамилии). Во многих случая авторизация на основе ролей предпочтительнее, поскольку ее гораздо легче реализовать. Если сотрудник безопасности должен будет сверять имя каждого гостя со списком допущенных, то процесс авторизации существенно замедлится. То же самое верно и для веб-приложений, хотя более вероятно, что роли будут следующими: менеджеры, администраторы, гости, продавцы, клиенты и т.д.

В веб-приложениях разные типы авторизации происходят на разных уровнях. Например, на самом верхнем уровне код может проверять идентичность пользователя и решать, можно ли продолжать данную операцию. На нижнем уровне можно настроить ASP.NET так, чтобы запрещать доступ к определенным веб-страницам или каталогам для определенных пользователей или ролей. На еще более низком уровне, когда код выполняет различные задачи — такие как подключение к базе данных, открытие файла запись в журнал событий и т.п. — операционная система Windows проверяет права учетной записи Windows, от имени которой выполняется данный код. В большинстве ситуаций вы не захотите полагаться на этот самый нижний уровень, поскольку код всегда будет выполняться от имени фиксированной учетной записи. В IIS 7.*x* это по умолчанию фиксированная учетная запись Network Service.

Существует несколько причин применения фиксированной учетной записи для запуска кода ASP.NET. Почти во всех приложениях права, выданные пользователю, не соответствуют правам, которые требуются приложению, работающему от имени пользователя. Как правило, код нуждается в более широком наборе привилегий, чтобы решать задачу идентификации, и вы не захотите выдавать такие права каждому пользователю, который может обращаться к вашему приложению. Например, коду может понадобиться создавать журнальные записи о возможных сбоях, даже если данному пользователю не разрешено напрямую записывать в журнал событий Windows, в файл или в базу данных. Аналогично приложения ASP.NET всегда должны иметь право доступа в каталог `c:\[КаталогWindows]\Microsoft.NET\Framework\[Версия]\Temporary ASP.NET Files`, чтобы создавать и кэшировать скомпилированные версии веб-страниц на машинном языке. И, наконец, может понадобиться использовать систему аутентификации, которая вообще никак не взаимодействует с Windows. Например, приложения электронной коммерции могут проверять адреса электронной почты пользователей по базе данных серверной стороны. В этом случае идентичность пользователя никак не соответствует пользовательской учетной записи Windows.

В некоторых редких случаях коду предоставляют возможность временно предполагать идентичность пользователя. Такой подход чаще всего используется при создании приложений ASP.NET для локальных сетей, в которых пользователи уже имеют строго определенные наборы привилегий Windows. В этом случае вам придется пополнить свой арсенал средств безопасности заимствованием прав, как упоминалось в предыдущем разделе и рассматривается в главе 22.

## Конфиденциальность и целостность

*Конфиденциальность* означает обеспечение невидимости данных для неавторизованных пользователей во время передачи их по сети или сохранении в хранилищах, таких как базы данных. *Целостность* — это обеспечение невозможности изменения

данных никем во время передачи по сети или сохранения в хранилище. И то, и другое основано на шифровании.

*Шифрование* — процесс кодирования данных, делающий невозможным их чтение другими пользователями. Шифрование в ASP.NET является средством, полностью отделенным от аутентификации, авторизации и заимствования прав. Его можно применять в комбинации с этими средствами либо самостоятельно.

Как упоминалось ранее, шифровать веб-приложения может требоваться по двум причинам.

- *Для защиты коммуникаций (передачи данных через сеть).* Например, вы хотите сделать невозможной кражу номеров кредитных карт, используемых в системе электронной коммерции, по открытым каналам Интернета. Стандартный подход к решению этой проблемы предусматривает применение SSL. Кроме того, SSL реализует цифровые подписи для обеспечения гарантии целостности. SSL не реализован ASP.NET, а является средством, предоставляемым IIS. Код веб-страницы (или веб-службы) не зависит от того, используется SSL или нет.
- *Для защиты постоянной информации (в базе данных или в файле).* Например, может понадобиться сохранить номер кредитной карты пользователя в базе данных для будущего использования. Хранение таких данных в виде простого текста в надежде на то, что веб-сервер не будет взломан, является плохой идеей. Вместо этого следует применять классы шифрования, которые предлагает .NET, и вручную шифровать данные перед их сохранением.

Не важно, что классы .NET, выполняющие шифрование, не связаны напрямую с ASP.NET. В действительности их можно использовать в любых приложениях .NET. Более подробно вопросы, связанные с шифрованием, цифровыми подписями и специальным шифрованием, рассматриваются в главе 25.

## Связываем все вместе

Итак, каким образом заставить работать вместе аутентификацию, авторизацию и заимствование прав в рамках веб-приложения?

Когда пользователи впервые посещают веб-сайт, они анонимны. Другими словами, ваше приложение не знает (и не заботится о том), кто они такие. Если вы не аутентифицируете их, все так и останется.

По умолчанию анонимные пользователи могут обращаться к любой странице ASP.NET. Но когда пользователь запрашивает веб-страницу, анонимный доступ к которой закрыт, выполняется несколько шагов (показанных на рис. 19.2).

1. Запрос отправляется веб-серверу. Поскольку идентичность пользователя в этот момент не известна, ему предлагается зарегистрироваться с использованием специальной веб-страницы или диалогового окна регистрации браузера. Специфические детали процесса регистрации зависят от типа применяемой аутентификации.
2. Пользователь предоставляет свое удостоверение, которое затем верифицируется — либо вашим приложением (в случае аутентификации с помощью форм), либо автоматически средствами IIS (в случае аутентификации Windows).
3. Если удостоверение пользователя подтверждается, ему предоставляется доступ к веб-странице. Если же оно оценивается как нелегитимное, ему предлагается повторить попытку регистрации, либо же выполняется переадресация на страницу с сообщением о закрытии доступа.

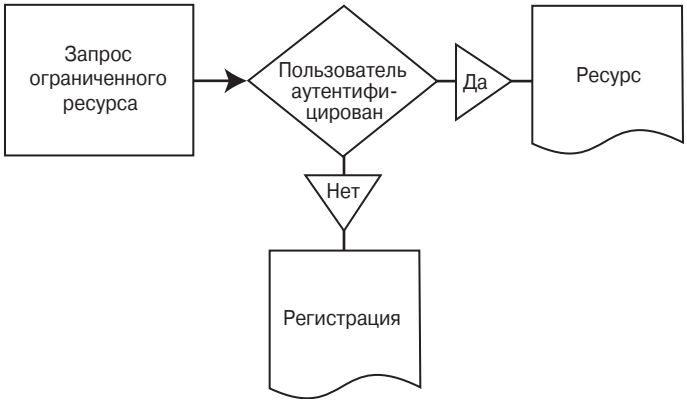


**Рис. 19.2.** Запрос веб-страницы, требующей аутентификации

Когда пользователь запрашивает защищенную веб-страницу, которая открыта только для определенных пользователей или ролей, процесс аналогичен, но добавляется дополнительный шаг (рис. 19.3).

1. Запрос отправляется веб-серверу. Поскольку идентичность пользователя в этот момент не известна, ему предлагается зарегистрироваться с использованием специальной веб-страницы или диалогового окна регистрации браузера. Специфические детали процесса регистрации зависят от типа применяемой аутентификации.
2. Пользователь предъявляет свое удостоверение, которое проверяется приложением. Это стадия аутентификации.
3. Удостоверение или роли аутентифицированного пользователя сравниваются со списком разрешенных пользователей и ролей. Если пользователь присутствует в списке, ему открывается доступ к ресурсу; в противном случае доступ будет закрыт.
4. Пользователи, которым отказано в доступе, либо приглашаются на повторную регистрацию, либо перенаправляются на веб-страницу с сообщением о закрытии доступа.

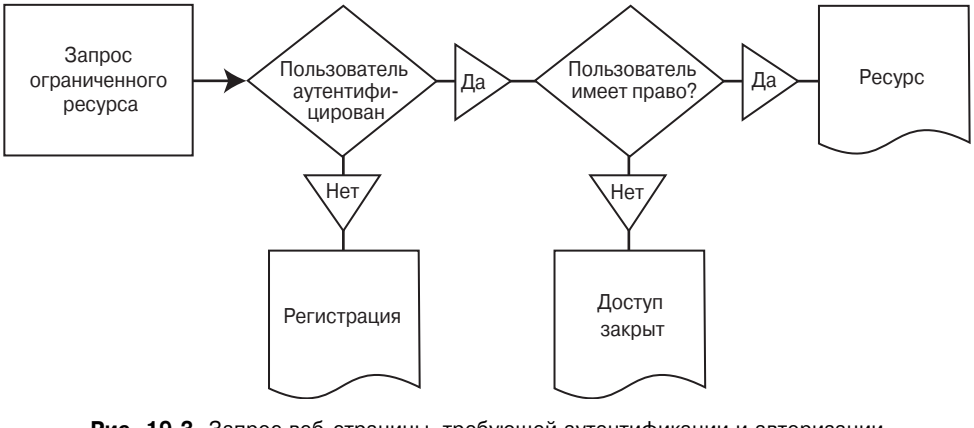


**Рис. 19.3.** Запрос веб-страницы, требующей аутентификации и авторизации

# Понятие SSL

Технология SSL (Secure Sockets Layer — уровень защищенных сокетов) позволяет шифровать коммуникации через HTTP. Протокол SSL поддерживается широким кругом браузеров и гарантирует, что передача информации между клиентом и веб-сервером не может быть расшифрована злоумышленниками. SSL необходим для сокрытия важной информации, такой как номера кредитных карт и конфиденциальные сведения внутреннего характера, но также важен и для аутентификации пользователей. Например, если вы создаете страницу регистрации, на которой пользователь отправляет свое имя и пароль, то должны применять SSL для шифрования этой информации. В противном случае злоумышленник сможет перехватить удостоверение пользователя и воспользоваться им для проникновения в систему.

Веб-сервер IIS предоставляет SSL как встроенную, готовую к использованию службу. Поскольку SSL работает под HTTP, его применение не изменяет способа работы с HTTP-запросами. Все шифрование и дешифрацию берут на себя функциональные средства SSL программного обеспечения веб-сервера (в данном случае — IIS). Единственное отличие состоит в том, что адрес, защищенный SSL, начинается с `https://`, а не `http://`. Трафик SSL также проходит через другой порт (обычно веб-серверы используют порт 443 для SSL-запросов и порт 80 — для обычных запросов).

Чтобы сервер поддерживал SSL-соединения, он должен иметь установленный сертификат X.509 (название X.509 было выбрано для соответствия стандарту каталогов X.500). Чтобы реализовать SSL, понадобится приобрести сертификат, установить его и соответствующим образом сконфигурировать IIS. Все эти шаги подробно описаны в последующих разделах.

## Понятие сертификатов

Прежде чем пересылать важные данные, клиент должен решить, можно ли доверять веб-сайту. Для этой достижения этой цели были спроектированы *сертификаты*, которые позволяют частично верифицировать идентичность пользователя. Сертификаты могут быть установлены на компьютере любого типа, но обычно они находятся на веб-серверах.

Организация приобретает сертификат у известного центра сертификации (Certificate Authority — CA) и устанавливает его на своем веб-сервере. Клиент доверяет CA и потому готов доверять информации сертификата, подписанного CA. Эта модель работает достаточно хорошо, т.к. маловероятно, чтобы злоумышленник решился на дополнительные расходы по приобретению и установке фальсифицированного сертификата. Центр сертификации также сохраняет информацию о каждом зарегистрированном пользователе. Однако наличие сертификата никоим образом не гарантирует надежность сервера, безопасность приложения или легитимность бизнеса. В этом смысле область действия сертификатов фундаментально ограничена.

Сам по себе сертификат содержит некоторую идентифицирующую информацию. Он подписывается защищенным ключом CA, что гарантирует его аутентичность и отсутствие модификаций. Промышленный стандартный сертификат, соответствующий x.509v3, включает следующую базовую информацию:

- имя, название организации и адрес держателя;
- открытый ключ держателя, который будет использоваться для реализации ключа SSL-сеансов для шифрования коммуникаций;
- даты проверки сертификата;
- серийный номер сертификата.

В дополнение сертификат также может включать специфичную для бизнеса информацию, такую как отрасль промышленности держателя, длительность присутствия его на рынке и т.п.

Двумя крупнейшими центрами сертификации являются:

- Thawte — `http://www.thawte.com`
- VeriSign — `http://www.verisign.com`

Если функция проверки идентичности CA не нужна (например, если сертификаты используются только в корпоративной сети), можете создать и пользоваться собственными сертификатами, настроив всех клиентов на доверие к ним. Для этого потребуется служба Active Directory и сервер Certificate Server (которые встроены в серверную операционную систему Windows). За более подробной информацией обращайтесь к специальным книгам по администрированию сетей Windows.

## Что собой представляет SSL

Как было описано в предыдущем разделе, каждый сертификат включает открытый ключ. Открытый ключ — это часть *асимметричной пары ключей*. Основная идея в том, что открытый ключ свободно предоставляется всем, кто в нем заинтересован. Соответствующий секретный ключ хранится под замком, и доступен только серверу. Трюк заключается в том, что все, что зашифровано одним из этих ключей, поддается расшифровке с помощью другого. Это значит, что клиент может получить открытый ключ и использовать его для шифрования секретного сообщения, которое может быть расшифровано с помощью соответствующего секретного ключа. Другими словами, клиент может создавать сообщения, которые сможет прочесть только сервер.

Этот процесс называется *асимметричным шифрованием*, и он является базовым строительным блоком SSL. Важный принцип асимметричного шифрования состоит в том, что реконструировать секретный ключ, анализируя соответствующий ему открытый ключ, невозможно. Сделать это было бы чрезвычайно дорого в плане вычислительных ресурсов (даже сложнее, чем взломать одно из зашифрованных сообщений). Однако асимметричное шифрование также имеет свои ограничения, а именно — оно намного медленнее и генерирует большие по объему сообщения, чем симметричное шифрование.

*Симметричное шифрование* — это разновидность шифрования, с которым интуитивно знакомо большинство людей. В нем один и тот же секретный ключ используется для шифрования и расшифровки сообщения. Недостаток симметричного шифрования в том, что оба участника должны знать секретное значение для того, чтобы осуществлять обмен сообщениями. Однако передавать эту информацию через Интернет нельзя, поскольку злоумышленники могут перехватить ее и в результате получить возможность расшифровывать все последующие сообщения. Главный трюк SSL заключается в том, что он комбинирует асимметричное и симметричное шифрование. Асимметричное шифрование используется только при начальной передаче ключа — другими словами, соглашения о секретном значении. Затем это секретное значение применяется для симметричного шифрования последующих сообщений, что гарантирует наилучшую из возможных производительность.

Весь процесс работает так, как описано ниже. При этом под *клиентом* понимается веб-браузер, выполняющийся на машине конечного пользователя, а под *сервером* — веб-сервер, обслуживающий веб-сайты, к которым пользователь желает получить доступ.

1. Клиент отправляет запрос на соединение с сервером.
2. Сервер подписывает свой сертификат и отправляет его клиенту. Это завершает фазу квитирования (“рукопожатия”) при обмене.

3. Клиент проверяет, издан ли сертификат CA, которому он доверяет. Если это так, он переходит к следующему шагу. В сценарии с веб-браузером клиент может предупредить пользователя сообщением о том, что он не распознал CA, и предложить пользователю принять решение о продолжении. Клиент распознает CA, когда его сертификат помещается в хранилище доверенных корневых центров сертификации (Trusted Root Certification Authorities) операционной системы. Сертификаты, находящиеся в этом хранилище, можно просмотреть в окне свойств Internet Explorer, щелкнув на кнопке Издатели в разделе Сертификаты вкладки Содержание.
4. Клиент сравнивает информацию в сертификате с информацией, присланной сайтом (включая доменное имя и его открытый ключ). Клиент также проверяет, допустим ли сертификат стороны сервера, не был ли он отменен и издан ли он заслуживающим доверия CA. Затем клиент принимает соединение.
5. Клиент сообщает серверу, какие ключи шифрования он поддерживает для коммуникаций.
6. Сервер выбирает наибольшую подходящую длину ключа и информирует клиента.
7. Используя указанную длину ключа, клиент случайным образом генерирует симметричный ключ шифрования. Он будет использован в период транзакции между сервером и клиентом. Это гарантирует оптимальную производительность, поскольку симметричное шифрование намного быстрее асимметричного.
8. Клиент шифрует ключ сеанса, используя открытый ключ сервера (из сертификата), и затем отправляет серверу зашифрованный ключ сеанса.
9. Сервер принимает зашифрованный ключ сеанса и расшифровывает его, используя свой защищенный ключ. После этого и клиент, и сервер имеют общий секретный ключ, и могут применять его для шифрования коммуникаций в период, пока длится сеанс.

Обратите внимание, что симметричный ключ генерируется случайным образом и используется только в период действия сеанса. Это ограничивает риск нарушения безопасности. Во-первых, сложнее взломать зашифрованные сообщения с применением криптоанализа, поскольку сообщения из других сеансов использоваться не могут. Во-вторых, даже если ключ будет вычислен злоумышленником, он останется действительным только на период существования одного сеанса.

Другой интересный момент связан с тем, что клиент должен генерировать симметричный ключ. Дело в том, что у клиента имеется открытый ключ сервера, который может быть использован для шифрования сообщений, доступных для чтения только серверу. У сервера нет соответствующей информации о клиенте, и потому он не может зашифровать сообщение. Это также означает, что если клиент предоставит ненадежный ключ, все взаимодействие окажется под угрозой. Например, в старых версиях браузера Netscape для создания симметричного ключа применялся слабый генератор случайных чисел. Это повышало вероятность угадывания ключей злоумышленниками.

При развертывании приложения, возможно, вы решите приобрести сертификаты от авторитетного CA, такого как VeriSign. Это, в частности, касается веб-сайтов и Интернет-браузеров, которые распознают ограниченное количество центров сертификации автоматически. Если, например, применяется тестовый сертификат для шифрования коммуникаций с защищенной частью веб-сайта, клиентский браузер отобразит предупреждение о том, что сертификат поставлен неизвестным CA.

## Конфигурирование SSL на IIS 7.*x*

Прежде всего, понадобится издать сертификат для веб-сервера. Для этого раскройте корневой узел веб-сервера в навигационном дереве консоли управления и выберите средство Server Certificates (Сертификаты сервера), как показано на рис. 19.4.


**Рис. 19.4.** Опция Server Certificates в IIS 7.*x*

После открытия детального представления консоль управления отобразит список сертификатов, установленных на сервере. Первый интересный момент в IIS 7.*x* связан с возможностью установки множества серверных сертификатов на одном веб-сервере, которые могут использоваться для различных сконфигурированных на нем веб-сайтов (рис. 19.5). Это — замечательное усовершенствование по сравнению со старыми версиями IIS, которые позволяли устанавливать только один серверный сертификат на веб-сервер.

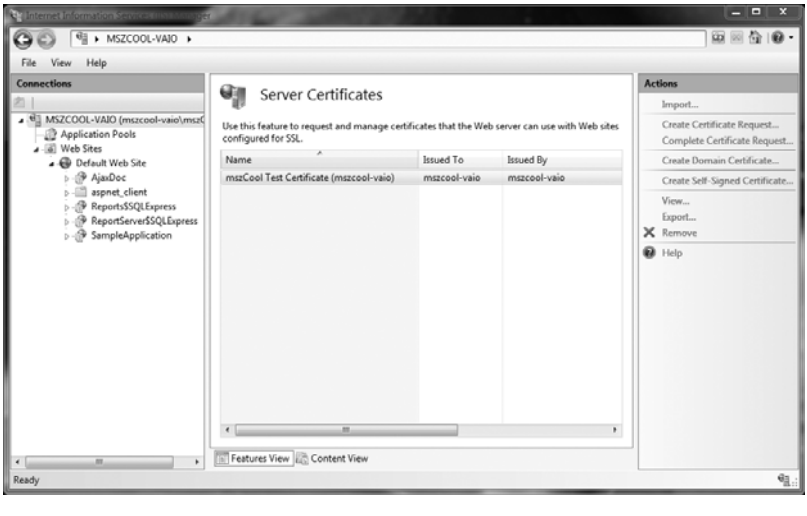


**Рис. 19.5.** Список серверных сертификатов, установленных на веб-сервере IIS 7.*x*

В детальном представлении средства Server Certificates панель задач в правой части консоли управления содержит необходимую задачу для установки серверных сертификатов. Она позволяет автоматически создать запрос сертификата, который можно применить для запроса нового сертификата у CA. Для создания нового запроса просто используется ссылка на задачу Create Certificate Request (Создать запрос сертификата) в панели задач, что позволит создать тот же закодированный в Base64 файл запроса для отправки его в запросе к CA. После получения сертификата от CA можно завершить выполняющийся запрос, щелкнув на ссылке Complete Certificate Request (Завершить запрос сертификата) в панели задач внутри средства Server Certificates консоли управления. Подобным образом можно запросить и сконфигурировать сертификат SSL для автономного веб-сервера. Если необходимо запросить сертификат для вашего собственного CA, можно воспользоваться мастером онлайнового центра сертификации (Online Certification Authority), щелкнув на ссылке Create Domain Certificate (Создать сертификат домена). Затем этот сертификат конфигурируется на вашем собственном CA и применяется для подписания изданных им сертификатов.

Честно говоря, этот процесс довольно запутанный, если вы — просто разработчик, которому нужно протестировать SSL на собственном веб-приложении. Поэтому IIS 7.*x* включает дополнительную опцию, которая изначально не была доступна в предыдущих версиях IIS: создание самоподписанного сертификата для вашей собственной машины. Все, что понадобится указать для самоподписанного сертификата — это дружественное имя, которое будет отображено в списке. Впоследствии мастер создает сертификат, применяя криптографические функции вашей машины, и установит этот сертификат на веб-сервер. Важно понимать, что такие сертификаты должны применяться только в целях тестирования, потому что ни один браузер кроме вашего, запущенного на вашей машине разработчика, не имеет понятия об этом сертификате, а потому выдаст предупреждение, что сертификат недействителен.

После конфигурирования и установки серверные сертификаты могут использоваться для SSL-коммуникаций внутри веб-сайтов, обслуживаемых вашим веб-сервером IIS. Для этой цели вам понадобится настроить привязки протокола для SSL, а также параметры SSL для веб-приложений внутри веб-сайтов.

### *Конфигурирование привязок для SSL*

Как было подробно описано в главе 18, привязки используются для открытия доступа к содержимому веб-сайтов через определенные протоколы, IP-адреса и порты. Хост-заголовки для доступа к нескольким веб-приложениям через один и тот же IP-адрес и порт также настраиваются в привязках. Чтобы применять SSL для приложений, сконфигурированных внутри веб-сайта, понадобится настроить привязку протокола SSL для веб-сайта. Для этого просто выберите веб-сайт (такой как Default Web Site (Веб-сайт по умолчанию)) в навигационном дереве диспетчера служб IIS и щелкните на ссылке Bindings (Привязки) в панели задач справа. Откроется диалоговое окно Web Site Bindings (Привязки сайта), которое позволяет сконфигурировать необходимые привязки. Добавьте новые привязки, чтобы обеспечить доступ к приложению через разные IP-адреса, порты и протоколы. Диалоговое окно Web Site Bindings показано на рис. 19.6.

Щелкнув на кнопке Add (Добавить), можно добавить новую привязку веб-сайта. Щелкнув на кнопке Edit (Изменить), можно модифицировать существующую привязку в списке. На рис. 19.7 показана конфигурация привязки для включения SSL на веб-сайте.

Легко заметить, что протокол сконфигурирован как `https`, запущенный на IP-адресе по умолчанию для сервера, с использованием порта 443 для SSL-доступа (который является портом по умолчанию для SSL). Далее в раскрывающемся списке в нижней части окна можно выбрать сертификат, используемый для трафика SSL на выбранном веб-сайте.

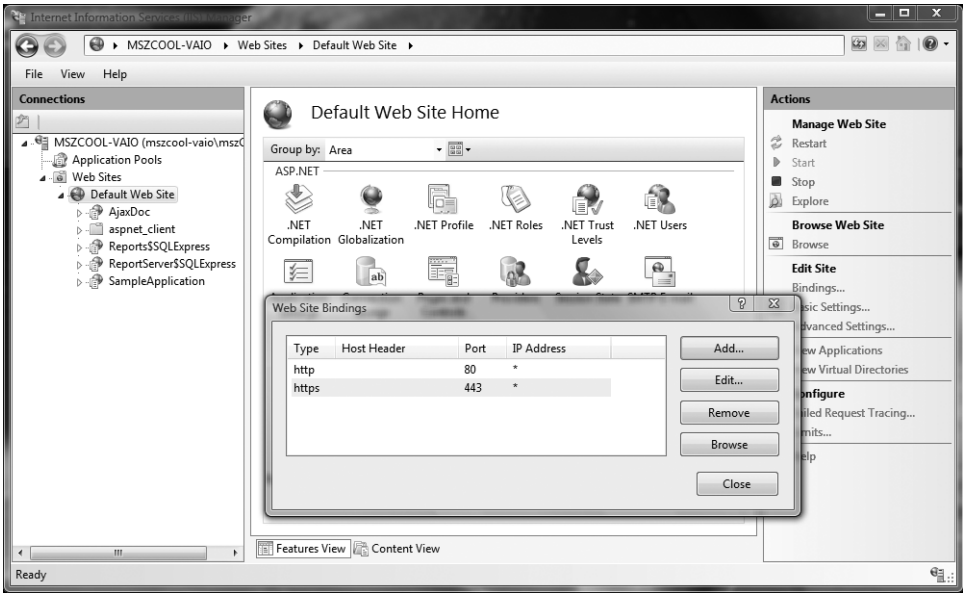


**Рис. 19.6.** Конфигурирование привязок для веб-сайта

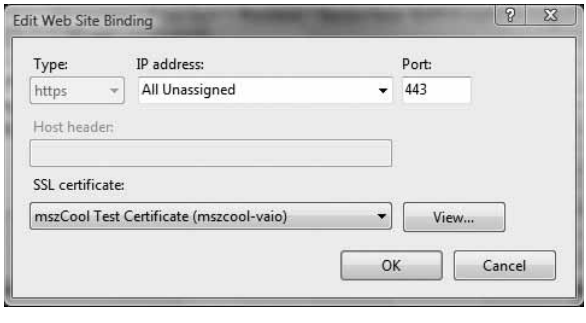


**Рис. 19.7.** Конфигурация привязки SSL на веб-сервере IIS 7.*x*

В этом списке доступен каждый установленный ранее сертификат, и для каждого веб-сайта на веб-сервере можно сконфигурировать свой сертификат. После конфигурирования привязки SSL для веб-сайта можно включить SSL для веб-приложений внутри веб-сайта.

### *Шифрование информации с помощью SSL*

Включение SSL в IIS конфигурируется на уровне веб-приложений. После настройки привязок на уровне веб-сайтов можно выбрать веб-приложение в навигационном дереве консоли управления IIS и активизировать конфигурацию средства SSL, как показано на рис. 19.8.

Здесь можно указать, требуется ли SSL-кодирование для выбранного веб-приложения, и нужны ли клиентские сертификаты для аутентификации пользователей. При использовании аутентификации с применением клиентского сертификата потребуется сконфигурировать отображение сертификатов на пользователей, чтобы они были в конечном итоге аутентифицированы IIS при извлечении определенного сертификата.

Эти отображения должны быть настроены в разделе `<iisClientCertificateMapping>` внутри раздела `<system.webServer>` конфигурационного файла `web.config`. За дополнительной информацией об отображении клиентских сертификатов обращайтесь к документации Microsoft, доступной в MSDN или TechNet по адресу:

```
http://msdn.microsoft.com/en-us/library/Aa347495.aspx
```

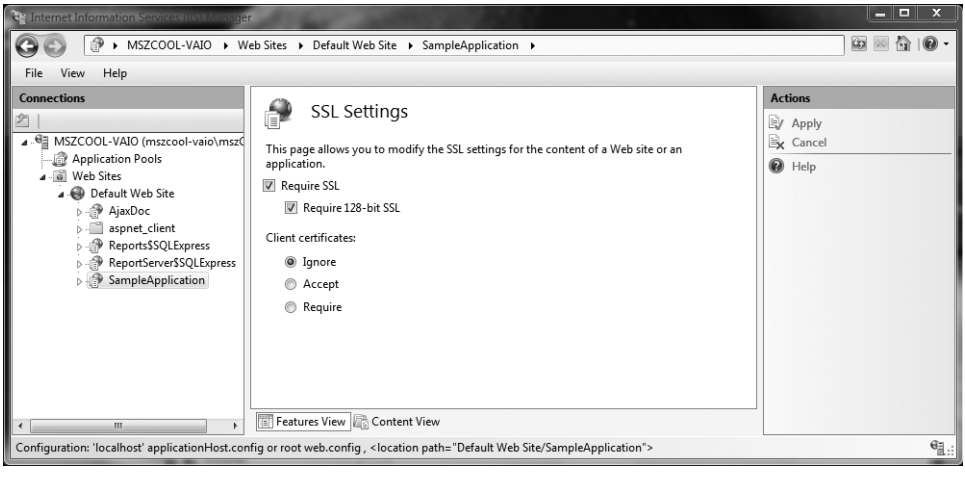


**Рис. 19.8.** Включение SSL-трафика на веб-сайте

# Резюме

Благодаря ASP.NET, программисты наконец-то получили согласованный и полный набор инструментов управления безопасностью. Как и со многими другими инструментальными средствами в мире ASP.NET, присутствие платформы безопасности просто означает, что вам придется выполнять меньше работы при реализации различных сценариев аутентификации и авторизации. Платформа ASP.NET предлагает два типа поставщиков аутентификации: аутентификацию Windows и аутентификацию с помощью форм. Вдобавок ASP.NET включает необходимые интерфейсы и классы, которые нужны для построения собственной системы аутентификации и авторизации. Все эти средства подробно рассматриваются в последующих главах.